Flask Templates

Rachit Joshi

Flask Templates

Flask templates allow developers to return HTML pages from Flask applications and separate application logic from webpage design. Flask uses the Jinja2 template engine to create dynamic and reusable HTML pages.

Why Use Flask Templates?

- Separate Python application logic from HTML presentation.
- Create dynamic web pages using data supplied by Flask view functions.
- Reuse HTML layouts and components more easily.
- Keep web applications organized and easier to maintain.

Returning an HTML Response

A Flask view can return HTML directly, although this can become difficult to maintain as a page grows.

```
from flask import Flask
app = Flask(__name__)

@app.route('/')
def message():
    return "<html><body><h1>Hi, welcome to the website</h1></body></html>"

if __name__ == '__main__':
    app.run(debug=True)
```

Rendering External HTML Files

Flask provides the render_template() function for rendering an external HTML file. Template files are normally stored in a folder named templates inside the application directory.

```
from flask import Flask, render_template

app = Flask(__name__)

@app.route('/')
def message():
    return render_template('message.html')

if __name__ == '__main__':
    app.run(debug=True)
```

Template Directory Structure

```
project/
│
├── script.py
└── templates/
    └── message.html
```

Flask looks for templates in the templates directory associated with the application.

Example HTML Template

```
<html>
<head>
    <title>Message</title>
</head>
<body>
    <h1>Hi, welcome to the website</h1>
</body>
</html>
```

Jinja2 Template Delimiters

Delimiter	Purpose
{% ... %}	Used for statements such as loops and conditions.
{{ ... }}	Used to print expressions or variable values.
{# ... #}	Used for template comments that are not included in the output.
# ... ##	Used for line statements in Jinja templates.

Displaying Dynamic Data

Variables can be passed from a Flask view function to a template and displayed with the {{ ... }} expression delimiter.

```
@app.route('/user/<uname>')
def message(uname):
    return render_template('message.html', name=uname)
```

```
<h1>Hi, {{ name }}</h1>
```

Embedding Python-Style Statements

Jinja template syntax allows control structures such as for loops to generate dynamic HTML. Statements are written using {% ... %}, while values can be displayed using {{ ... }}.

```
<h2>Printing table of {{ n }}</h2>
{% for i in range(1, 11) %}
    <h3>{{ n }} X {{ i }} = {{ n * i }}</h3>
{% endfor %}
```

Using Static Files

Flask applications can serve static resources such as CSS and JavaScript from a static directory. Templates can use url_for() to create the appropriate URL for a static file.

```
<link rel="stylesheet"
      href="{{ url_for('static', filename='css/style.css') }}">
```

Example Project Structure with Static Files

```
project/
├── script.py
├── templates/
│   └── message.html
└── static/
    └── css/
        └── style.css
```

Benefits of Flask Templates

- Improves separation between application logic and presentation.
- Makes dynamic HTML generation easier.
- Supports reusable templates and organized project structures.
- Makes it easier to work with CSS and JavaScript static files.
- Helps create maintainable Flask applications.

Important Points to Remember

- Flask uses the Jinja2 template engine.
- HTML templates are normally stored in the templates folder.
- render_template() is used to render an external template.
- {{ ... }} is used to display expressions or variables.
- {% ... %} is used for template statements.
- {# ... #} is used for template comments.
- Static CSS and JavaScript files can be stored in the static folder.

Conclusion

Flask templates provide an organized way to build dynamic HTML pages in Python web applications. By using render_template(), the templates directory, Jinja2 expressions and statements, and static resources, developers can separate application logic from presentation and create maintainable Flask applications.